Quick
Reference
Guide

HP 18320A
DataCommC
Programming
Language

r the HP 4954A Protocol Analyzer

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_05846

HP 4954A Protocol Analyzer

HP 18320A DataCommC Programming Language

Quick Reference Guide

Manual Part Number: 18320-99403
Microfiche Part Number: 18320-98815

Printed in U.S.A. August, 1990
E0890

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

If your software application or hardware should fail, contact your local Hewlett-Packard Sales Office listed in the protocol analyzer operating manual.

©Copyright 1989, 1990 Hewlett-Packard Company.
Colorado Telecommunications Division
5070 Centennial Boulevard
Colorado Springs, CO, 80919-2497

Contents

Directories and Files

Directories 1
File Types 1
Files 2
Application Modules 3

Language Elements

Keywords and Reserved Words 4
C Statements 4
Preprocessor Directives 4
Data Types 4
Escape Sequences 5
Arithmetic Operators 5
Assignment Operators 6
Bitwise Operators 6
Logical Operators 6
Relational Operators 7
Unary Operators 7
Operator Precedence 8

Editor

Cursor Control Keys 9
Edit Mode Keys 10
Command Mode Keys 12

Library Functions

Alphabetical Listing 13

Printing History

New editions are complete revisions of the manual. Update packages are issued between editions. They contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change when an update is incorporated.

Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correlation between product updates and manual updates.

Edition 1	February 1989
Edition 2	September 1990

Directories and Files

Directories

The basic, multi-level DataCommC directory structure is created when the application is installed on the hard disc. DataCommC files are also installed at this time (see "Files"). You can select directories by either entering the path from DataCommC or by using the HP 4954A Protocol Analyzer Mass Store menu `Select Directory` feature.

The installation process creates directories at analyzer directory levels 1, 2, and 3. The following list defines the directories at each level and their case sensitive path.

Level 1	Level 2	Level 3
c:C	c:C/Decode	
	c:C/Include	
	c:C/Help	
	c:C/HP	
	c:C/Lib	
	c:C/Manager	
	c:C/System	c:C/System/Bin
		c:C/System/Source
	c:C/User	
c:Term		

File Types

The file types supported by the analyzer, DataCommC, the terminal emulator, and the HP 4954A Disc Utilities, Revision 02.00, are as follows:

68000source	events	listing
applicmodule	ext.rundata	macro
archive	extendedmenu	menus
backup	include	menus&data
bufferdata	isdnintrfc	object
corruptfile	isdnprotocol	program
csource	library	system
datacode	linkcommand	text
directory	linkstats	user
errorlog	linksymbol	

Directories and Files

Files

The basic DataCommC files are stored in the directories created when you install the application. The following list defines the paths for these files.

c:C/System/Bin/
- Assembler.Program
- Compiler.Program
- Drivers.Applicmodule
- ExamEvents.System
- FrontEnd.Program
- Linker.Program
- Make.Program
- RUNDISC.Program
- TESTER.Program
- TIMER.Program

c:C/System/Source/
- Compiler.Text
- Linker.Text
- MAKEFILE.Text

c:C/Include
- ascii.Include
- assert.Include
- conio.Include
- ctype.Include
- decode.Include
- dlib.Include
- ebcdic.Incl
- fcntl.Include
- leads.Include
- math.Include
- message.Include
- retval.Include
- sec_ch.Include
- setjmp.Include
- stdio.Include
- stdlin.Include
- system.Include
- time.Include
- video.Include

c:C/Decode/
- BOPS.System
- BOPS.Program
- QLLC_DEC.Program
- QLLC_DEC.System
- QLLC_DEC.Text
- S_SHELL.Program
- SNA_DEC.Program
- SNA_DEC.System
- SNA_DEC.Text
- SNASAVE.Text
- X25_DEC.Program
- X25_DEC.System
- X25_DEC.Text

c:C/Lib/
- DataCommC.Library

c:Term/
- Term.Text

c:C/Manager
- help.System
- S_SHELL.System
- SHELL.System
- TESTER.System

Application Modules

DataCommC specific application modules are stored on the HP 4954A Protocol Analyzer when you install the application. They are selected from the analyzer or from a program. The DataCommC application modulesand their storage locations are:

c:
- C_manager
- Term_Emul

c:/Term
- Drivers
- Term_Emul

c:C/Help
- FIDS.Text
- help.Program
- HINTS.Text
- X25HELP.Text

c:C/System/Bin
- C_develop
- Drivers

c:autoload
- C_Manager

Language Elements

Keywords and Reserved Words

auto	do	for	return	typedef
break	double	goto	short	union
case	else	if	sizeof	unsigned
char	enum	int	static	void
continue	extern	long	struct	while
default	float	register	switch	

C Statements

;[1]	continue	for	if	switch
break	do	goto	return	while

Preprocessor Directives

#asm	#endasm	#if	#ifndef	#line
#define	#endif	#ifdef	#include	#undef
#else				

Data Types

char	short	signed short int
double	short int	unsigned
enum	signed char	unsigned char
float	signed int	unsigned int
long	signed long	unsigned long
long double	signed long int	unsigned long int
long int	signed short	unsigned short int

[1] Empty Statement

Language Elements

Escape Sequences

Note: The screen drivers only interpret `\n`.

`\"`	Double quotation mark
`\'`	Single quotation mark
`\\`	Backslash
`\b`	Backspace
`\`*ddd*	Octal notation (*ddd*) for one ASCII character
`\f`	Form feed character
`\n`	Newline character
`\r`	Carriage return character
`\t`	Horizontal tab character
`\v`	Vertical tab character
`\x`*ddd*	Hexadecimal notation (*ddd*) for one ASCII character

Arithmetic Operators

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder
++	Increment
--	Decrement

Language Elements

Assignment Operators

Operator	Function
+ =	Add to value and assign.
- =	Subtract from value and assign.
* =	Multiply by value and assign.
/ =	Divide by value and assign.
% =	Get remainder from division and assign.
=	Assignment.
& =	AND assignment.
^ =	Exclusive OR assignment.
| =	Inclusive OR assignment.
> > =	Shift the value to the right and assign.
< < =	Shift the value to the left and assign.

Bitwise Operators

Operator	Function
&	AND
^	Exclusive-OR
|	Inclusive-OR
~	One's complement
< <	Shift Left
> >	Shift Right

Logical Operators

Operator	Function
&&	AND
||	OR
!	Logical NOT

Language Elements

Relational Operators

Operator	Function
a < b	Is a less than b?
a > b	Is a greater than b?
a == b	Is a equal to b?
a != b	Is a not equal to b?
a <= b	Is a less than or equal to b?
a >= b	Is a greater than or equal to b?

Unary Operators

Operator	Function
!	Logical NOT
~	One's complement
-	Arithmetic negation
*	Indirection
&	Address of
+	Unary plus

Operator Precedence

The precedence for executing operators in a statement are:

- The operator precedence level.
- The evaluation order.

The following operators are listed in -

- Precedence level sequence from the highest to the lowest.
- There is no order of precedence within a sequence other than the evaluation order, which is Left to Right, unless otherwise stated.

Operator	Type (Evaluation Order)
( ) [ ] . ->	Expression
- ~ ! * & + + – – sizeof (type)	Unary (Right to Left)
* / %	Multiplicative
+ -	Additive
<< >>	Shift
< > <= >=	Relational (inequality)
== !=	Relational (equality)
&	Bitwise AND
^	Bitwise-exclusive OR
|	Bitwise-inclusive OR
&&	Logical AND
||	Logical OR
? :	Conditional (Right to Left)
= *= /= %= += -= <<= >>= &= |= ^=	Simple and compound assignment (Right to Left)
,	Sequential Evaluation

Editor

ursor Control Keys

ressing **Develop Programs** and **Edit** activates the Text Editor. The nalyzer keyboard primarily controls Edit Mode functions while Text Editor softkeys ontrol Command Mode functions. The editor **Escape** switches between these odes and within levels of the Command Mode. The cursor control keys have milar, but not identical functions, in both modes.

unction	Key
love the cursor down one line.	▼
love the cursor down almost one window (i.e., wards the end of the file buffer).	Next page
love the cursor left one character.	◀
love the cursor left to the start of the line.	Shift ◀
love the cursor left to the start of the previous ord.	CTRL ◀
love the cursor right one character.	▶
love the cursor right to the end of the line.	Shift ▶
love the cursor right to the start of the next word.	CTRL ▶
love the cursor up one line.	▲
love the cursor up almost one window (i.e., towards ie start of the file buffer).	Prev page
love the cursor to the first character in the file uffer.	Home or Shift Roll up
love the cursor after the last character in the file affer.	Shift Home or Shift Roll down

Editor

Function	Key
Scroll the screen down one line.	Shift ▼ or Roll down
Scroll the screen up one line.	Shift ▲ or Roll up
Select the cursor display mode by toggling between: ▪ Inverse video (default). ▪ Inverse, flashing video. ▪ Underscore. ▪ Inverse, half-bright video.	Shift Back space

Edit Mode Keys

Copy a block of text into the work buffer.	
1. Position the cursor at the start of the text block and press	Escape D M
2. Position the cursor at the end of the text block and press	Escape B C
Recall a block of text from the work buffer. Position cursor at point to insert text and press	Escape B I
Delete the character at the cursor position.	Delete char
Delete the character to the left of the cursor.	Back space
Delete the characters from the cursor position to the end of the line.	Clear line
Delete the current line including the ^R and the ^F.	Shift Delete char
Edit in two windows.	Escape W Z
Exit the Editor without saving the file.	Escape F Q
Exit the Editor after saving the file.	Escape F S

Function	Key
In INSERT mode, insert an ASCII tab character in the buffer or insert the number of spaces to the next tab stop and move the cursor.	Tab
In INSERT mode, insert a ^{C}R and a ^{L}F in the file buffer.	Return
Insert a new line before the current line.	Shift + Insert char
In REPLACE mode, move the cursor to the next tab stop.	Tab
In REPLACE mode, move the cursor the start of the next line.	Return
Set fixed tabs.	Escape D T F *number of spaces*
Toggle between UPPERCASE and lowercase characters.	Caps lock
Toggle between the Insert and Replace modes.	Insert char
Transfer to Command Mode.	Escape
Write text to another file.	Escape F W

Command Mode Keys

Move the cursor left one keyword.	
Move the cursor left to the first keyword.	Home or Shift + Roll up
Move the cursor right one keyword.	
Move the cursor right to the last keyword	Shift + Home or Shift + Roll down
Select the highlighted keyword.	Return
Return to next higher command mode level.	Escape
Transfer to Edit mode.	Escape

```
nt abs (num)
nt num;
```

Returns the absolute value of *num*.

```
#include <math.h>
double acos (x)
double x;          /* a radian value between -1 and 1 */
```

Returns the arccosine of *x*.

```
#include <math.h>
double asin (x)
double x;          /* a radian value between -1 and 1 */
```

Returns the arcsine of *x*.

```
#include <assert.h>
ssert (expr)
nt expr;
```

Inclusion of this macro in a program identifies logic errors by evaluating *expr*. A FALSE (0) result prints a diagnostic message in the format:

Assertion failed: expr, file *ffff*.csource, line l*nnn*

where:

ffff	Source file path and name.
nnn	Assert statement line number.

To exclude **assert** ( ) functions from the preprocessor created source file, enter `#define NDEBUG` before the `#include <assert.h>` directive.

```
#include <math.h>
louble atan (x)
louble x;          /* a radian value between -1 and 1 */
```

Returns the arctangent of *x*.

```
#include <math.h>
ouble atan2 (x,y)
ouble x,y;         /* radian values between -1 and 1 */
```

Returns the arctangent of *x/y*. *x* and *y* can be given opposite signs.

```
#include <stdlib.h>
double atof(str)
char *str;
```

Returns the double value of the character string pointed to by *str*. The string can include leading blanks and tabs, an optional sign (+ -), a string of digits with an optional decimal point, and an optional e or E followed by optionally signed integer.

```
#include <stdlib.h>
int atoi(str)
char *str;
```

Returns the integer value of the character string pointed to by *str*. The string can include leading blanks and tabs, an optional sign (+ -), and a string digits.

```
#include <stdlib.h>
long atol(str)
char *str;
```

Returns the long integer value of the character string pointed to by *str*. The string can include leading blanks and tabs, an optional sign (+ -), and a string of digits.

```
void beep()
```

Causes the analyzer to generate a half-second beep.

```
void bufcpy(dest,source,n)
char *dest;
char *source;
int n;
```

Copies *n* bytes from *source* to *dest*, regardless of control characters.

```
#include <stdlib.h>
char *calloc(num,size)
int num;
int size;
```

Allocates *num* contiguous blocks of zeroed memory of *size* bytes. Returns a pointer to the memory allocated; NULL if memory cannot be allocated.

```
#include <math.h>
double ceil(x)
double x;
```

Returns the next double larger than *x*.

```
void center(string, row)
char *string;
int row;                /* 1 - 25 */
```

Writes a centered string of characters on the designated row. *string* is padded with leading and trailing spaces to center. The cursor is moved to row 1, column 1. No action is taken when *row* is greater than 25, or when *string* is NULL.

```
void ch_dir(path)
char *path;
```

Changes the current directory of the calling process to *path*.

```
int clear_timer(timer_number)
int timer_number;
```

Cancels **set_timer( )** for the DataCommC timer *timer_number*.

```
void clearn(n, ptr)
int n;
char *ptr;
```

Zeros *n* bytes beginning at *ptr*.

```
#include <fcntl.h>
int close(file_desc)
int file_desc;
```

Closes the file defined by the pointer *file_desc* previously returned by **open( )**.

```
#include <math.h>
double cos(x)
double x;               /* in radians */
```

Returns the cosine of *x*.

```
#include <math.h>
double cosh(x)
double x;
```

Returns the hyperbolic cosine of *x*.

```
#include <math.h>
double cotan(x)
double x;
```

Returns the cotangent of *x*.

```
void cursor_off()
```

Makes the cursor invisible.

```
void cursor_on()
```

Makes the cursor visible.

```
int decode_exit()
```

Deletes the decode program currently in memory.

```
#include <message.h>
#include <conio.h>
int decode_keys_on()
```

Notifies the decode program that it has control of the softkeys and routes any keyboard inputs to the decode program.

```
int decode_loaded()
```

Returns whether a code program is loaded.

```
int decode_spawn(decode_program)
char *decode_program;
```

Loads the decode program specified by *decode_program* into memory.

```
#include <message.h>
#include <conio.h>
int decode_start(run_time, screen_size, share_keyboard)
int run_time;          /* 0 (examine buffer mode), 1 (run mode) */
int screen_size;       /* valid range 3 through 21 */
int share_keyboard;    /* 0 (exclusive), 1 (shared) */
```

Starts the decode program currently loaded in memory.

```
int decode_stop( )
```

Stops a running decode program. The program remains in memory.

```
#include <message.h>
int display_keys_on( )
```

Notifies the Run Time Display (RTD), spawned by **start_display( )** with *share_keyboard* enabled, that it has control of the keyboard and can display its softkeys.

On exiting, the RTD will notify its parent process that it is now relinquishing control of the keyboard by sending a DISPLAY message with a KEYS_OFF subtype.

```
int eq_file_time(file_1, file_2)
char *file_1;
char *file_2;
```

Returns 0 if the last modified times are equal, -1 if the last modified time of *file_1* is earlier than that of *file_2*, and 1 if the last modified time of *file_1* if after that of *file_2*. A -2 is returned if one or both of the files are not found.

```
#include <message.h>
void error_notification(error_msg, error_subtype)
char *error_msg;
int error_subtype;   /* 0 (NON_FATAL_ERROR), 1 (FATAL_ERROR) */
```

Sends an error message to error_QID.

```
#include <system.h>
int examine_data (path_name, priority)
char *path_name;   /* decode program to run, NULL if default */
int priority;      /* 1 - 199, default is 100 */
```

Spawns the DataCommC decode process; the keyboard is not shared and the entire screen is used. The default program is the one selected in the C Manager's "Buffer Manager" screen.

```
void exit (exit_val)
int exit_val;
```

Kills the calling process and returns *exit_val* to the parent process.

```
#include <math.h>
double exp (x)
double x;
```

Returns the exponential function of x.

```
#include <math.h>
double fabs (x)
double x;
```

Returns the absolute value of x.

```
#include <stdio.h>
int fclose (file)
FILE *file;
```

Closes a file opened with `fopen( )`.

```
#include <stdio.h>
int fflush (file)
FILE *file;
```

Flushes the buffer associated with a file.

```
#include <stdio.h>
int fgetc (file)
FILE *file;
```

Gets the next character from a file.

```
#include <stdio.h>
char *fgets (str, n, file)
char *str;
int n;
FILE *file;
```

Returns a pointer to string *str* of up to *n* - 1 characters read from *file*.

```
int file_time (name, year, month, day, hour, min, sec)
char *name;
char year[3];
char month[3];
char day[3];
char hour[3];
char min[3];
char sec[3];
```

Returns the ASCII time string (year, month, day, hour, min, sec) file was last modified.

```
void filln (n, ptr, fill)
int n;
char *ptr;
char fill;
```

Fills *n* bytes, starting at *ptr*, with *fill*.

```
#include <math.h>
double floor (x)
double x;
```

Returns the largest double integer not greater than *x*.

```
#include <stdio.h>
FILE *fopen(name, access_mode)
char *name;
char *access_mode;
    /* "a"  Append in write mode.
     * "a+" Append in read and write mode.
     * "r"  Read operation.
     * "r+" Read and write operation.
     * "w"  Write operation. The contents of an existing file are destroyed.
     * "w+" Write and read operation. An existing file is destroyed.
     */
```

Opens *name* in the specified *access_mode* and returns a pointer to the assigned FILE structure, or returns NULL if the file could not be opened.

```
int fprintf(file, format, arg)
FILE *file;
char *format;
```

Prints a formatted string of characters to *file* based on the conversion specifications in *format*. Refer to **printf( )** for information on data conversion.

```
#include <stdio.h>
int fputc(ch, file)
int ch;
FILE *file;
```

Writes the character *ch* to *file*.

```
#include <stdio.h>
int fputs(str, file)
char *str;
FILE *file;
```

Writes the string *str* to *file*. The terminal NULL character is not written.

```
#include <stdio.h>
int fread(ptr, size, n, file)
char *ptr;
int size;          /* size of each element in bytes. */
int n;             /* number of elements. */
FILE *file;
```

Reads up to *n* elements of *size* characters from *file* and stores the information in the buffer pointed to by *ptr* and returns the actual count of items.

```
#include <stdlib.h>
void free(ptr)
char *ptr;
```

Releases the block of memory previously allocated by `malloc( )` or `calloc( )`.

```
int fscanf(file, format, args)
FILE *file;
char *format;
```

Reads a character string from *file* and converts it using the conversion information in *format* placing the results in the args locations include in the function call. Refer to **scanf( )** for information on data conversion.

```
#include <stdio.h>
int fseek(ptr, offset, origin)
FILE *ptr;
long offset;
int origin;          /* SEEK_SET (start of file),
                        SEEK_CUR (current file position),
                        SEEK_END (end of file) */
```

Move the file pointer for *ptr* from *origin* by *offset* bytes.

```
#include <stdlib.h>
void ftoa(val, buf, precision, type)
double val;          /* the number to be converted. */
char *buf;           /* the buffer to receive the ASCII string */
int precision;       /* the number of digits right of the decimal point. */
int type;            /* the format (see printf( )):
                        0 = e format
                        1 = f format
                        2 = g format */
```

Converts a floating-point or double-precision number to an ASCII character string.

```
#include <stdio.h>
int fwrite(ptr, size, n, file)
char *ptr;
int size;          /* size of each element in bytes. */
int n;             /* number of elements. */
FILE *file;
```

Writes *n* elements of *size* characters to *file* from buffer *ptr*.

```
#include <video.h>
int get_attribute( )
```

Returns the current video attributes (NORMAL, INVERSE, BLINK, or HALF_BRIGHT) for each of the character set types (ASCII, HEX, or EBCDIC). The SPECIAL and UNDERLINE attributes may also be set for graphics characters.

```
int get_bitrate(bitrate)
long *bitrate;
```

Returns the current bit rate in the memory location specified by *bitrate*.

```
#include <dlib.h>
int get_channelconfig(channel, config)
char channel;      /* 1 (DTE), 2 (DCE), 4 (SDTE), 5 (SDCE) */
char *config;      /* 0 (NONE), 1 (RX), 2 (TX) */
```

Returns the current configuration of *channel* in the memory location specified by *config*.

```
void get_cursor(row, column)
int *row, *column;
```

Returns current cursor row and column positions *row* and *column*.

```
#include <dlib.h>
int get_data_source(data_source, data_file)
char *data_source; /* 2 (LINE_RUN), 3 (DISC_RUN) */
char *data_file;
```

Returns the current values for *data_source* and *data_file*.

```
#include <dlib.h>
int get_datacode (data_code)
int *data_code;      /* 1 (ASCII8), 2 (ASCII7), 3 (EBCDIC), 4 (HEX8)
                        5 (HEX7), 6 (HEX6), 7 (HEX5), 8 (USER_DEF) */
```

Returns the current value of *data_code*.

```
#include <dlib.h>
#include <see_ch.h>
int get_dce_channel ( )
```

Returns the channel currently being driven by the DCE: 1 (PRIMARY) or 2 (SECONDARY).

```
char *get_dir (path_name)
char path_name [82];
```

Returns the default directory drive/directory/subdirectory path character string in *path_name*.

```
#include <dlib.h>
#include <sec_ch.h>
int get_dte_channel ( )
```

Returns the channel number currently being driven by the DTE: 1 (PRIMARY) or 2 (SECONDARY).

```
#include <dlib.h>
int get_duplex (duplex)
int *duplex;         /* HDX (1), FDX (2) */
```

Returns the value of duplex value in *duplex*.

```
#include <dlib.h>
int get_errorcheck (error_check)
int *error_check;    /* 1 (CRC_16), 2 (CRC_CCITT),
                        3 (NO_ERROR_CHECK), 4 (CRC_12),
                        5 (LRC), 6 (CRC_6) */
```

Returns the value of the current error check type in *error_check*.

```
int get_event_bounds (num_events, first_event, last_event)
void **first_event, **last_event;
int *num_events;
```

Returns pointers to the chronologically first and last events, and the number of events, currently in the event buffer, or zeroes if the buffer is empty.

```
#include <leads.h>
int get_ignored_leads (lead_id_map)
long *lead_id_map;
```

Returns in *lead_id_map* the current value for the mask defining the leads to be ignored by the data link control hardware.

```
#include <leads.h>
int get_lead (lead_id, lead_state, message_ptr)
char lead_id;
char lead_state;     /* 1 (ON), 0 (OFF), 0x0FF (DONT_CARE) */
MESSAGE *message_ptr;
```

Returns 0 (SUCCESSFUL) if the message pointer *message_ptr* is a DataCommC message with a lead status of *lead_state* for the lead specified in *lead_id*.

```
#include <dlib.h>
int get_lead_control (lead_control)
int *lead_control;   /* 2 (LEAD_CONTROL_DEFAULT,
                        1 (LEAD_CONTROL_USER_DEF */
```

Returns the current value for *lead_control*.

```
#include <dlib.h>
int get_mode (mode)
int *mode;           /* 1 (ASYNC_1), 2 (ASYNC_1_5), 3 (ASYNC_2),
                        4 (SYNC_DCE), 5 (SYNC_NRZI), 6 (SYNC_DTE) */
```

Returns the current synchronization mode value in *mode*.

```
#include <isdn.h>
int get_op_mode ( )
```

Returns the current event handling mode: 0 (MANUAL_RELEASE), 1 (AUTO_RELEASE).

```
#include <dlib.h>
int get_parity (parity)
int *parity;          /* 1 (ODD_PARITY, 2 (EVEN_PARITY), 3 (NO_PARITY),
                         4 (IGNORE) */
```

Returns the current parity in *parity*.

```
long get_pri_bitrate ( )
```

Returns the bit rate of the primary channel.

```
#include <dlib.h>
int get_protocol (protocol)
int *protocol;        /* 1 (BSC), 2 (SDLC), 3 (HDLC), 4 (X25), 5 (COPS),
                         7 (X75) */
```

Returns the current protocol value in *protocol*.

```
#include <dlib.h>
int get_resyncafter (num_after)
int *num_after;
```

Returns in *num_after* the current value for the number of characters after a BCC that the analyzer processes before it drops and then establishes synchronization.

```
#include <dlib.h>
int get_resynchars (itext, otext1, otext2, otext3, otext4, otext5, otext6)
char *itext, *otext1, *otext2, *otext3, *otext4, *otext5, *otext6;
```

Returns the current resynchronization character values for both in text (*itext*) and out of text (*otext* through *otext6*).

```
#include <dlib.h>
int get_resyncmode (rmode)
int *rmode;           /* Manual (1), Auto (2) */
```

Returns in *rmode* the current resynchronization mode value.

```
long get_sec_bitrate ( )
```

Returns the bit rate of the secondary channel.

---

long **get_sec_frame_size** ( )

Returns the character size selected for the secondary channel.

---

long **get_sec_mode** ( )

Returns the mode selected for the secondary channel.

```
#include <dlib.h>
int get_startbcc (startbcc1, startbcc2)
char *startbcc1, *startbcc2;
```

Returns the current start BCC character values in *startbcc1* and *startbcc2*.

```
#include <dlib.h>
#include <leads.h>
int get_stop_states (lead_id, stop_state)
long lead_id;
int *stop_state;      /* 0 (OFF), 1 (ON), 2 (NOT_DRIVEN), 3 (MARKING),
                         4 (ACTIVE) */
```

Returns in *stop_state* the stop state value specified for the *lead_id* lead.

```
#include <dlib.h>
int get_stopbcc (stopbcc1, stopbcc2)
char *stopbcc1, *stopbcc2;
```

Returns the current stop BCC character values in *stopbcc1* and *stopbcc2*.

```
#include <dlib.h>
int get_synchars (smode, syncchar1, syncchar2)
int *smode;
char *syncchar1, *syncchar2;
```

Returns the current synchronization character values in *syncchar1* and *syncchar2* and the current mode value in *smode*.

```
#include <time.h>
void get_time (time)
struct tm *time;
```

Writes the current real time clock values to structure *time* that includes:

element	range
tm_msecs	0 - 900 milliseconds (100 ms intervals).
tm_secs	0 - 59 seconds.
tm_min	0 - 59 minutes.
tm_hour	0 - 23 hours.
tm_day	1 - 31 days.
tm_mon	1 - 12 months.
tm_year	0 - 99 years (1900 - 1999).

```
#include <dlib.h>
int get_transtext (trans_mode, ttextchar)
int *trans_mode;
char *ttextchar;
```

Returns the transparent text character in *ttextchar* and the mode in *trans_mode*.

```
#include <stdio.h>
int getc (file)
FILE *file;
```

Reads a character from *file*.

```
#include <conio.h>
int getch ( )
```

+Reads an unbuffered *character* from stdin.

```
#include <stdio.h>
int getchar ( )
```

Reads a buffered *character* from stdin.

```
char *gets(buffer)
char *buffer;
```

Reads a string of characters from stdin into *buffer*. The read is terminated with a new line character, but the new line character is not included in *buffer*.

```
#include <message.h>
int hold_event(event)
void *event;
```

Retains the event message *event* previously sent to another processes message queue with **send_message( )** by incrementing the events held count.

```
void init_rs232( )
```

Initializes the analyzer serial port with **init_rs232( )** prior to calling any RS-232C function or **print_char( )**.

```
int init_trigger(message_type,number_of_subtypes,subtype_args)
unsigned message_type;  /* in message.h */
unsigned num_subtypes;  /* range: 1-31;
                           0: clear all triggers for message_type
                           >31: sets all subtype for message_type
                           to triggers and ignore subtype_args */
unsigned subtype_args;
```

Initializes trigger_on_message to trigger on *message_type* and *subtype_args*.

```
int is_key_avail( )
```

Returns TRUE if the calling process' key queue contains a character.

```
int is_msg_avail( )
```

Returns TRUE if the calling process' message queue contains a message.

```
#include <ctype.h>
int isalnum(ch)
int ch;
```

Returns TRUE if the character is an alphanumeric character (A - Z, a - z, or 0 - 9).

```
#include <ctype.h>
int isalpha(ch)
int ch;
```

Returns TRUE if the character is an alphabetic character (A - Z or a - z).

```
#include <ctype.h>
int isascii(ch)
int ch;
```

Returns TRUE if the character is an ASCII character (0x00 - 0x7F).

```
#include <ctype.h>
int iscntrl(ch)
int ch;
```

Returns TRUE if the character is a control character (0x00 - 0x1F, or 0x7F).

```
#include <ctype.h>
int isdigit(ch)
int ch;
```

Returns TRUE if the character is a numeric digit (0 - 9).

```
#include <ctype.h>
int isgraph(ch)
int ch;
```

Returns TRUE if the character is a printable character, excluding the space character (0x21 - 0x7E).

```
#include <ctype.h>
int islower(ch)
int ch;
```

Returns TRUE if the character is a lowercase alphabetic character (a - z).

```
#include <ctype.h>
int isprint(ch)
int ch;
```

Returns TRUE if the character is a printable character (0x20 - 0x7E).

```
#include <ctype.h>
int ispunct(ch)
int ch;
```

Returns TRUE if the character is a punctuation character (0x21 - 0x2F, 0x3A - 0x40, 0x5B - 0x60, 0x7B - 0x7E).

```
#include <ctype.h>
int isspace(ch)
int ch;
```

Returns TRUE if the character is a white space character (0x09 - 0x0D, or 0x20).

```
#include <ctype.h>
int isupper(ch)
int ch;
```

Returns TRUE if the character is an uppercase alphabetic character (A - Z).

```
#include <ctype.h>
int isxdigit(ch)
int ch;
```

Returns TRUE if the character is a hexadecimal digit (A - F, a - f, or 0 - 9).

```
#include <stdlib.h>
char *itoa(num,str,radix)
int num;
char *str;
int radix;              /* The base of the converted argument */
```

Returns in *str* the string representation of the integer *num*.

void **keyboard_lock** ( )

Locks the keyboard for the exclusive use of the calling process. The keyboard remains locked until the calling process unlocks it or the calling process terminates. Processes that call this function with the keyboard already locked are blocked until the first process relinquishes the lock.

The functions **getc** ( ), **getch** ( ), **gets** ( ), **scanf** ( ), **ungetc** ( ), and **ungetch** ( ) implicitly call this function. **fgetc** ( ), **fgets** ( ), **fscanf** ( ), and **fread** ( ) call this function when the file pointer is stdin, as does **read** ( ) if the file handle is zero.

---

int **keyboard_unlock** ( )

Unlocks the keyboard if issued by the calling process that locked the keyboard.

```
#include <math.h>
double log(x)
double x;
```

Returns the natural logarithm of *x*.

```
#include <math.h>
double log10(x)
double x;
```

Returns the base 10 logarithm of *x*.

```
#include <setjmp.h>
void longjmp(env, val)
jmp_buf env;
int val;                /* must be none zero */
```

Restores the stack environment values from *env* that were saved by **setjmp** ( ) and causes program execution to continue as if **setjmp** ( ) returned *val*.

```
#include <stdio.h>
long lseek(handle,offset,origin)
int handle;
long offset;
int origin;          /* SEEK_SET (start of file),
                        SEEK_CUR (current file position),
                        SEEK_END (end of file) */
```

Moves the file pointer for file *handle* from *origin* by *offset* bytes.

```
#include <stdlib.h>
char *ltoa(num,str,radix)
long num;
char *str;
int radix;           /* The base of the converted argument */
```

Returns in *str* the string representation of the long *num*.

```
char *make_file(path_name,size)
char *path_name;
unsigned long size;
```

Creates a new file of the length *size* bytes.

```
char *makedir(rtn_path,dev,path,file_name,file_type)
char *rtn_path[82];
char dev;
char *path;
char *file_name;
char *file_type;
```

Returns in *rtn_path* the complete pathname of the file specified by *dev*, *path*, *file_name* and *file_type*.

```
#include <stdlib.h>
char *malloc(size)
unsigned int size;
```

Returns a pointer to a block of memory of *size* bytes..

```
#include <stdio.h>
#include <fcntl.h>
int open(filename,access)
char *filename;
int access;          /*O_APPEND (add text at end of file)
                       O_CREAT (create a new file)
                       O_RDONLY (read a file)
                       O_RDWR (read and write file)
                       O_WRONLY (write a file)
                       O_EXCL (return an error if O_CREAT name is
                       existing file) */
```

Opens the file *filename* in *access* mode and returns a file handle for the open file.

```
void parsedir(name,dev,path,file_name,file_type);
char *name;
char dev[3];
char path[56];
char file_name[11];
char file_type[14];
```

Parses the complete pathname for file *name* into its components and returns the strings in *dev*, *path*, *file_name*, and *file_type*.

```
#include <math.h>
double pow(x,y)
double x,y;
```

Returns x raised to the power of y.

```
void print_char(chr)
char chr;
```

Writes the character *chr* to the RS-232C port.

int **printf**(*format, arg*)
char **format*;

Prints a string of characters to standard output formatted by the conversion specification included in *format*. For each conversion specification there must be an argument element.

Each conversion specification is in the form:

%[flag][width][.precision][l]format

where:

flag — Specifies the field justification.

- Left justify output; the default is to right justify.

width — Specifies the minimum number of characters used for conversion.

A "*" for the width parameter indicates an integer argument specifies the width.

precision — Specifies the maximum number of digits output from the string, or the maximum number of digits to the right of the decimal point.

A "*" for the precision parameter indicates an integer argument specifies the precision.

l — Modifies the normal input length as a long; used with d, o, and x.

format — c a single character
- d a signed decimal
- e scientific notation

f a floating point number
- g uses d, e, or f, whichever is shorter
- o an unsignedoctal integer

s a character string
- u an unsigneddecimal integer
- x an unsignedhexadecimal integer

```
#include <video.h>
void put_all_sks(s1, s2, s3, s4, s5, s6, s7, s8, attribute)
char *s1, *s2, *s3, *s4, *s5, *s6, *s7, *s8;
int attribute;          /* normally INVERSE and/or HALF_BRIGHT */
```

Writes labels to the specified softkeys; a NULL string keeps the existing label. *attribute* applies to all softkeys. Each label starts on upper softkey row. Each softkey row can contain a maximum of nine characters; labels greater than nine characters are truncated. The single opening quotation mark (') marks the start of the lower row.

```
#include <video.h>
#include <conio.h>
void put_sk(label, softkey_id, attribute)
char *label;
int softkey_id;         /* SK1, SK2,..., SK8 */
int attribute;          /* normally INVERSE and/or HALF_BRIGHT */
```

Writes the *label* to *softkey_id* with the specified *attribute*. The **put_all_sks**( ) softkey characteristics also apply to **put_sk**( ).

```
#include <stdio.h>
int putc(ch, file)
int ch;
FILE *file;
```

Writes the character *ch* to *file*.

```
void putch(ch)
char ch;
```

Writes the character *ch* to at current cursor position with the current display attributes and then advances the cursor position horizontally.

```
#include <stdio.h>
int puts(string)
char *string;
```

Writes the string *string* starting at the current cursor position with the current display attributes. If the display is not in transparent text mode, the cursor is move to column 1 of the next row.

int **rand** ( )

Returns a positive pseudo-random number between 0 and 32,767. The **srand** ( ) function can be used to initialize a random starting point before calling **rand** ( ).

```
#include <stdio.h>
int read (fdesc, buff, cnt)
int fdesc;
char *buff;
unsigned int cnt;
```

Reads up to *cnt* bytes from file *fdesc* and returns the number of characters returned in *buff*

```
#include <message.h>
int read_message (message, wait, timeout);
MESSAGE *message;
int wait;              /* 0 (WAIT) for timeout
                          1 (NO_WAIT), return immediately */
long timeout;          /* 0 (WAIT_FOREVER), wait indefinitely */
```

Returns *message* from the message queue. A NULL return indicated a successful read.

```
#include <dlib.h>
int read_pod_id (pod_id)
int *pod_id;           /* 1 (RS232), 2 (V_35), 3 (RS449), 4 (X_21),
                          5 (MIL188C), 8 (NO_POD) */
```

Returns the current pod identification in *pod_id*.

```
void read_rs232 (ch)
char *ch;
```

Reads a character from the RS-232C port into *ch*. Use **init_rs232** ( ) first.

```
#include <video.h>
void read_vidram (row, column, character, attribute)
int row;              /* 1 - 25 */
int column;           /* 1 - 80 */
char *character;
int *attribute;
```

Reads the *characterh* and *attribute* from video RAM at *row* and *column*.

```
#include <message.h>
int release_event (event)
void *event;
```

Releases access to *event*. Memory may not be usable until all chronologically earlier events are also released.

```
int resend_bops (ptr, n, length)
char *ptr;
unsigned int n;
unsigned int length;
```

Retransmits a bit-oriented protocol frame previously formatted and transmitted with a call to **sendf**( ). Using **resstartbops**( ) is faster than calling **sendf**( ) twice.

If *ptr* is specified, the first *n* characters of the original frame are replaced with the character string referenced by *ptr*. If *n* is zero, no replacement occurs. The *length* parameter specifies the new frame length. The original frame is send unchanged if *length* is zero.

```
int reset_pod( )
```

Resets the attached pod by placing all leads in either tri-state or not driven states and resets all pod hardware.

```
void restore_cursor( )
```

Restores the cursor position saved with **save_cursor**( ).

```
int resync( )
```

Causes the analyzer to drop sync when Protocol is COPs, the data link control hardware is Running, and Resync Mode is Manual (set by **set_resync**( )).

---

void **roll_down** (*startrow*, *startrow*, *num_lines*)
int *startrow*, *startrow*, *num_lines*;

Rolls screen between rows *startrow* and *startrow* down *num_lines* lines

---

void **roll_up** (*startrow*, *startrow*, *num_lines*)
int *startrow*, *startrow*, *num_lines*;

Rolls screen between rows *startrow* and *startrow* up *num_lines* lines

---

void **rs232_lock** ( )

Locks the RS-232C port for the calling process. The calling process is blocked until it can lock the port. Use **init_rs232** ( ) first.

---

int **rs232_read_ready** ( )

Returns TRUE if there is a character to be read, otherwise FALSE is returned. Use **init_rs232** ( ) first.

---

void **rs232_unlock** ( )

Releases the RS-232C port for use by another process. Use **init_rs232** ( ) first.

---

int **rs232_write_ready** ( )

Returns TRUE if a character can be transmitted, otherwise FALSE is returned. Use **init_rs232** ( ) first.

---

void **save_cursor** ( )

Saves the current cursor position for later retrieval by **restore_cursor** ( ).

nt **scanf** (*format*, *args*)
char **format*;

Reads a string of characters from the keyboard, and converts them using the conversion specification in *format*, and places the results in argses corresponding to the conversion specifications.

The *format* string can contain:

- conversion specifications.
- optional whitespace characters (tab, space, newline).
- optional alphanumeric characters (not part of whitespace or conversion specifications).

Each conversion specification is in the form:

%[flag][width][l]format

where:

flag	Can be an asterisk "*" that suppresses the assignment of the scanned data.
width	Specifies the maximum number of characters input.
l	Modifies the normal input length as a long.
format	ca single character d a signed decimal e a floating point number fa floating point number o an unsignedoctal integer sa character string u an unsigneddecimal integer x an unsignedhexadecimal integer

#include <message.h>
nt **send_message** (*QID*, *message*, *options*);
unsignedlong *QID*;
MESSAGE **message*;
nt *options*; /* ZERO (no additional action)
EXPEDITE (expedite message delivery) */

Sends a message to the QID identified queue.

```
#include <dlib.h>
int sendf(format, args)
char *format;
char *arg;
```

Sends a formatted string to the communications channel. As each string element is read, the character specification it contains is used to modify the corresponding field in the argument list.

The conversion specification format is:

%[dimension]converter

where:

dimension is an optional field that can be a constant indicating (1) the number of argument list fields, (2) the field length, or (3) the argument value, or an asterisk (*) indicating the value is included as an argument in the *arg* list.

converter is one of the following:

- b — a pointer to a buffer of raw data; the dimension field specifies the buffer length.
- c — the channel to use; 1 for primary and 2 for secondary.
- f — the frame check sequence (FCS) value (ABORT_FCS, BAD_FCS, GOOD_FCS, NO_FCS) for each %f. When no %f is included, the default value is GOOD_FCS.
- n — a series of one or move 8-bit values specified by the dimension field.
- s — an ASCII string using the current data code and parity; the dimension field specifies the absolute string length to include NULL characters. When there is no dimension field the string length equals the number of characters before the first NULL character.

```
#include <video.h>
void set_attribute(attribute)
int attribute;          /* NORMAL, INVERSE, BLINK, HALF_BRIGHT,
                           UNDERLINE, SPECIAL */
```

Sets the default video attribute type for the ASCII, hexadecimal, or EBCDIC character sets. Constants are OR'd together to establish the attribute value.

```
#include <dlib.h>
int set_bcc (startbcc1, startbcc2, stopbcc1, stopbcc2)
char startbcc1, startbcc2, stopbcc1, stopbcc2;
```

Establishes the protocol rules for calculating the block check character (BCC) for the COPS protocol with an error check selected. BCC accumulation occurs between *startbcc1* and *stopbcc1* or *startbcc2* and *stopbcc2*.

```
#include <dlib.h>
int set_bitrate (bitrate)
long bitrate;          /* 10 b/s - 256 Kb/s */
```

Sets the bit rate for transmitting and receiving data. If *bitrate* is less than the lower bound, the lower bound is used, and if it is more than the upper bound, the upper bound is used. Nominal performance is not guaranteed above 100 Kb/s Changes must be made before using **startdata** ( ).

```
#include <dlib.h>
int set_buffer_sizes (DTE_size, DCE_size)
int DTE_size, DCE_size;
```

Sets the DTE and DCE raw data buffer sizes (DTE_size, DCE_size), prior to **startdata** ( ), as a percentage of total buffer space. DataCommC defaults these settings to 50% when **set_buffer_sizes** ( ) is called after **startdata** ( ).

```
#include <dlib.h>
int set_channelconfig (channel, config)
char channel, config;
```

Establishes the transmit and receive configuration (*config*) of a DTE, DCE, SDTE, SDCE, PRIM_ALL, or ALL channel as TX, RX, or NONE. Only one channel at a time can be configured for TX. When the channel parameter is PRIM_ALL or ALL the configuration must be RX.

```
void set_cursor (row, column)
int row;               /* 1 - 25 */
int column;            /* 1 - 80 */
```

Positions the cursor at the specified *row* and *column*.

```
#include <dlib.h>
int set_data_source(data_source,data_file)
char data_source;  /* LINE_RUN, DISC_RUN */
char *data_file;
```

Sets the data buffer input source as either LINE_RUN or a DISC_RUN (i.e buffer data, menus and data, extended run). The default data source is specified in the Setup Menu. To use DISC_RUN call **set_data_source( )** before calling **startdata( )**. Ignore *data_file* if data_source is LINE_RUN. **startdata( )** opens *data_file*, and **stop_data( )** closes *data_file*.

```
#include <dlib.h>
#include <sec_ch.h>
int set_dce_channel(pri_sec)
int pri_sec;          /* 1 (PRIMARY), 2 (SECONDARY) */
```

Sets the channel to be driven by the DCE.

```
#include <video.h>
void set_disp_bank(disp_bank)
int disp_bank;        /* 0x000 (DISP_BANK0) ... 0x0300 (DISP_BANK3) */
```

Selects one of four character display banks. This function only has an effect after calling **set_attribute**(SPECIAL).

```
#include <dlib.h>
#include <sec_ch.h>
int set_dte_channel(pri_sec)
int pri_sec;          /* 1 (PRIMARY), 2 (SECONDARY) */
```

Sets the channel to be driven by the DTE.

```
#include <dlib.h>
int set_duplex(duplex)
int duplex;           /* HDX (1), FDX (2) */
```

Sets the communications mode for the DTE and DCE.

```
int set_error_handler(error_QID)
unsignedlong error_QID;   /* 0  Use default system error handler */
```

Designates *error_QID* as the NON_FATAL error queue.

```
nt set_ignored_leads (lead_id_map)
unsignedlong lead_id_map;
```

Sets ignore lead mask to *lead_id_map*. The data link control hardware will ignore transitions on leads that are mapped to a bit set in the mask.

```
#include <leads.h>
#include <dlib.h>
nt set_lead (lead_id, lead_state)
unsignedlong lead_id;
unsignedint lead_state; /* 0 (OFF), 1 (ON), 2 (NOT_DRIVEN) */
```

Set the lead defined by *lead_id* to *lead_state*.

```
#include <dlib.h>
nt set_lead_control (lead_control)
nt lead_control;    /* 2 (LEAD_CONTROL_DEFAULT),
                       1 (LEAD_CONTROL_USER_DEF) */
```

Sets the source of control information to *lead_control*.

```
#include <isdn.h>
nt set_op_mode (value)
nt value;           /* 0 (MANUAL_RELEASE), 1 (AUTO_RELEASE) */
```

Designates whether events are sent to the calling process' message queue or whether they are automatically released.

```
nt set_pri_bitrate (rate)
ong rate;           /* 10 b/s - 256 Kb/s */
```

Sets the bit rate of the primary channel.

```
#include <dlib.h>
int set_protocol(protocol,mode,datacode,parity,error_check)
int protocol;        /* 1 (BSC), 2 (SDLC), 3 (HDLC), 4 (X25), 5 (COPS),
                        7 (X75) */
int mode;            /* 1 (ASYNC_1), 2 (ASYNC_1_5), 3 (ASYNC_2),
                        4 (SYNC_DCE), 5 (SYNC_NRZI), 6 (SYNC_DTE)
                        0xFFF (NO_CHANGE) */
int datacode;        /* 1 (ASCII8), 2 (ASCII7), 3 (EBCDIC), 4 (HEX8),
                        5 (HEX7), 6 (HEX6), 7 (HEX5), 8 (USER_DEF),
                        0xFFF (NO_CHANGE) */
int parity;          /* 1 (ODD_PARITY), 2 (EVEN_PARITY), 3 (NO_PARITY),
                        4 (IGNORE), 0xFFF (NO_CHANGE) */
int error_check;     /* 1 (CRC_16), 2 (CRC_CCITT),
                        3 (NO_ERROR_CHECK), 4 (CRC_12), 5 (LRC),
                        6 (CRC_6), 0xFFF (NO_CHANGE) */
```

Sets the protocol and associated data communication parameters.

```
#include <dlib.h>
int set_resync(rmode,num_after,itext,otext1,otext2,otext3,otext4,otext5,
otext6)
int rmode;           /* 1 (MANUAL), 2 (AUTO) */
int num_after;
char itext,otext1,otext2,otext3,otext4,otext5,otext6;
```

For a character-oriented protocol using either the SYNC_DCE or SYNC_DTE synchronization modes, defines the rules for dropping synchronization. When *rmode* is AUTO, this is based on the number characters (*num_after*) when in (*itext*) or outside of transparent text mode (*otext1* through *otext6*). When *rmode* is MANUAL a call to **resync**() is required.

```
void set_rows(min_row,max_row)
int min_row,max_row;
```

Specifies the first (*min_row*) and last (*max_row*) rows, the screen area, or window, to which you can write data.

```
#include <video.h>
void set_screen_mode(screen_mode)
int screen_mode;   /* ASCII_DISPLAY (normal),
                      TRANSPARENT_DISPLAY (display all) */
```

Sets the screen mode to a normal display (ASCII_DISPLAY) or a special character display (TRANSPARENT_DISPLAY), where white space and non-printable characters do not change the display, but appear as a. For example, a line feed does not advance the cursor to the next line of text but displays LF.

```
long set_sec_bitrate(sec_bitrate)
long sec_bitrate;   /* 10 b/s - 19.2 Kb/s */
```

Sets the bit rate for the secondary channel.

```
long set_sec_frame_size(sec_frame_size)
int sec_frame_size; /* 5, 6, 7, or 8 */
```

Sets the number of bits in a secondary channel frame.

```
long set_sec_mode(sec_mode)
int sec_mode;       /* 1 (ASYNC_1), 2 (ASYNC_1_5), 3 (ASYNC_2) */
```

Sets the number of stop bits on the secondary channel.

```
#include <leads.h>
int set_stop_states(lead_id, stop_state)
unsigned long lead_id;
int stop_state;
```

Sets the state of the clock and control interface leads on the transmit line after calling **stop_data( )**.

```
#include <dlib.h>
int set_sync(sync_mode, syncchar1, syncchar2)
int sync_mode;
char syncchar1, syncchar2;
```

For character-oriented protocols, or SYNC_DTE or SYNC_DCE mode, defines the rules for resynchronizing (i.e., dropping and restoring synchronization). To define the synchronization character framing for creating events from the data stream, set *sync_mode* to ON. The actual frame is defined by *syncchar1* and *syncchar2*.

```
#include <time.h>
void set_time(time)
struct tm *time;
```

Sets the real-time clock system time from the structure *time* whose elements are:

element	range
tm_msecs	0 - 900 milliseconds (100 ms intervals).
tm_secs	0 - 59 seconds.
tm_min	0 - 59 minutes.
tm_hour	0 - 23 hours.
tm_day	1 - 31 days.
tm_mon	1 - 12 months.
tm_year	0 - 99 years (1900 - 1999).

```
int set_timer(number_of_tenths)
unsigned long number_of_tenths;
```

Sets a DataCommC timer for the *number_of_tenths* (0.1 seconds) duration a returns the non-negative timer number. When the time elapses, a messages i placed in the calling process' message queue.

```
#include <dlib.h>
int set_transtext(trans_mode, ttextchar)
int trans_mode;      /* ON, OFF */
char ttextchar;
```

Defines the transparent text mode protocol rules for a COPS protocol. Whe *trans_mode* is ON, *ttextchar* signals an entry into or an exit from the transparent text mode. While in the transparent text mode all control characters are treated as data and their functions are not executed.

```
#include <setjmp.h>
int setjmp(env)
jmp_buf env;
```

Stores the processor register values in *env* to be restored later by **longjmp( )**

```
#include <math.h>
double sin(x)
double x;              /* in radians */
```

Returns the sine of *x*.

```
#include <math.h>
double sinh(x)
double x;              /* in radians */
```

Returns the hyperbolic sine of *x*.

```
#include <system.h>
int spawn(path_name,number_of_user_args,user_args,priority,child_QID)
char *path_name;
int number_of_user_args;
char *user_args[];
int priority;          /* 1 - 199 */
long *child_QID;
```

Creates a process using system default characteristics that executes independently of the calling process. *child_QID* is the queue number of a successfully spawned child process.

```
#include <system.h>
int spawnw(path_name,number_of_user_args,user_args,priority,value)
char *path_name;
int number_of_user_args;
char *user_args[];
int priority;          /* 1 - 199; default 100 */
int *value;
```

Creates a process using system defaults that executes independently of the calling process. The calling program is suspended until the spawned process terminates. Arguments can be passed to the spawned programs via *number_of_user_args* and *user_args*. *value* is the value returned by the spawned process when it exits.

```
int sprintf(str,format,arg)
char *str;
char *format;;
```

Prints a formatted string of characters to memory location *str* based on the conversion specifications in *format*. Refer to **printf( )** for information on data conversion.

```
#include <math.h>
double sqrt(x)
double x;
```

Returns the square root of *x*.

```
void srand(val)
int val;
```

Sets the seed *val* for the pseudo-random number generator.

```
int sscanf(buffer,format,args)
char *buffer;
char *format;
```

Reads a character string from memory location *buffer* and converts it using th conversion information in *format* placing the results in the args locations include in the function call. Refer to **scanf( )** for information on data conversion.

```
int start_data( )
```

Causes the start data acquisition. The data source is set by the Setup menu unless **set_data_source( )** is used. The leads are left in there previous state when data acquisition begins. The data buffer is flushed. Incoming data ever are stored in the event buffer.

```
int start_display(file,screen_size,share_keyboard)
char *file;
int screen_size;      /* 3 through 21 */
int share_keyboard;   /* 0 Don't share, 1 Share */
```

Initializes and spawns a Run Time Display (RTD) or decode. *file* is the full path name of the protocol display program. The RTD always has a lower priority (priority level 2) than any user process.

```
int stop_data( )
```

Stops data acquisition and event formation by halting protocol data link cont hardware.

nt **stop_display** ( )

Terminates the Real Time Display (RTD).

```
#include <string.h>
char *strcat (s2, s1)
char *s2;
char *s1;
```

Concatenates *s1* to the end of *s2*. Does not test for *s2* overflow.

```
#include <string.h>
char *strchr (str, ch)
char *str;
char ch;
```

Returns a pointer to the first occurrence of character *ch* in string *str*, or NULL if not found.

```
nt strcmp (s1, s2)
char *s1;
char *s2;
```

Uses a lexigraphic comparison to compare strings *s1* and *s2*:

s1 < *s2*	return a negative number (-1).
s1 = *s2*	return zero (0).
s1 > *s2*	return a positive number (1).

```
nt strcmpi (s1, s2)
char *s1;
char *s2;
```

Uses lexigraphic comparison to compare *s1* and *s2*. Comparison is independent of upper or lower case characters.

s1 < *s2*	return a negative number (-1).
s1 = *s2*	return zero (0).
s1 > *s2*	return a positive number (1).

```
#include <string.h>
char *strcpy(s2,s1)
char *s1;
char *s2;
```

Copies *s1* to *s2*.

```
int strlen(str)
char *str;
```

Returns the length of the string *str*.

```
#include <string.h>
char *strncat(s2,s1,n)
char *s2;
char *s1;
int n;
```

Copies a maximum of *n* characters from *s1* to the end of *s2*.

```
int strncmp(s1,s2,n)
char *s1;
char *s2;
int n;
```

Uses lexigraphic comparison to compare the first *n* characters of *s1* and *s*

s1 < *s2*	returns a negative number (-1).
s1 = *s2*	returns a zero (0).
s1 < *s2*	returns a positive number (1).

```
#include <string.h>
char *strncpy(s2,s1,n)
char *s1;
char *s2;
int n;
```

Copies *n* characters from *s1* to *s2*. When *s1* contains less than *n* charac
s2 is padded with NULL characters.

```
nclude <string.h>
ar *strrchr(str,ch)
ar *str;
ar ch;
```

Returns a pointer to the last occurrence of the character *ch* in the string *str*, or a NULL if *ch* not found.

```
nclude <string.h>
ar *strsave(str)
ar *str;
```

Allocates memory for the string pointed to by *str*, then copies *str* into the allocated memory and returns a pointer to the new copy, or NULL.

```
nclude <string.h>
ar *strtrm(string)
ar *string;
```

Trims trailing spaces from the string pointed to by *string*.

```
id sys_msg(string)
ar *string;
```

Centers *string* on row 23 of the display, padding *string* with white space. Only the first 80 characters are written. The cursor position is preserved.

```
nclude <math.h>
uble tan(x)
uble x;          /* in radians */
```

Returns the tangent of x.

```
nclude <math.h>
uble tanh(x)
uble x;          /* in radians */
```

Returns the hyperbolic tangent of x.

```
#include <ctype.h>
int toascii(c)
char c;
```

Converts a single character *c* to an ASCII character by converting the high order bit to zero. Does not change *c* if it is already an ASCII character.

```
#include <ctype.h>
int tolower(c)
char c;
```

Converts an uppercase character *c* to a lowercase character.

```
#include <ctype.h>
int toupper(c)
char c;
```

Converts a lowercase character *c* to an uppercase character.

```
#include <message.h>
int trigger_on_message(message)
MESSAGE *message;
```

Reads messages from the process queue *message*. Returns on either if the Message type and subtype equal the setup called by **init_trigger**( ) or subty > 127.

```
#include <stdio.h>
void ungetc(ch,file)
int ch;
FILE *file;
```

Places the character *ch* back into *file*.

```
#include <conio.h>
int ungetch(ch)
int ch;
```

Places the keystroke *ch* back into the input buffer.

```
int unlink(file)
char *file;
```

Deletes the file *file*.

```
int wait(number_of_tenths)
unsigned long number_of_tenths;    /* tenths of a second */
```

Suspends the calling process for the time specified by *number_of_tenths*.

```
#include <message.h>
int wait_data(data_type, timeout, data_event)
int data_type;
long int timeout;
MESSAGE *data_event;
```

Gets the first *data_type* message in the calling process' message queue during the period *timeout* (milliseconds).

```
#include <leads.h>
int wait_lead(lead_id, lead_state, timeout, lead_event);
long lead_id;
int lead_state;        /* 1 (ON), 0 (OFF), 0x0FF (DONT_CARE) */
unsigned long timeout;
MESSAGE *lead_event;
```

Scans the message queue and collects the first lead change message that matches *lead_id* and *lead_state* until *timeout*, in milliseconds, occurs. The message pointer is returned in *lead_event*. Other events are discarded.

```
void wipe(row_1, row_2)
int row_1, row_2;
```

Clears the screen from *row_1* through *row_2*.

```
#include <stdio.h>
int write(fdesc, buff, cnt)
int fdesc;
char *buff;
unsigned int cnt;
```

Writes *cnt* bytes of *buff* to the file *fdesc*.

---

void **write_rs232** (*ch*)
char *ch*;

Writes the character *ch* to the RS-232C port. Use **init_rs232** ( ) first.

#include <video.h>
void **write_vidram** (*row*, *column*, *character*, *attribute*)
int *row*, *column*;
int *attribute*;
char *character*;

Writes *character*, with *attribute*, to the *row* and *column* video RAM locatio
The results are not effected by **set_rows** ( ) and text can be printed outside o
those text areas.

Copyright © 1990
Hewlett-Packard
Printed in USA 9/9

Reorder No. or
Manual Part No
18320-99403

18320-99403E0990